CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE143: COMPUTER CONCEPTS & PROGRAMMING

Chapter – 10

Structures

DEPSTAR

Devang Patel Institute of Advance Technology and Research

Objectives

- To explain the basic concepts of structure
- To process a structure
- To manipulate data using a struct
- To Learn about the relationship between a struct and functions
- To discover how arrays are used in a struct
- To create an array of struct items
- To explain the concept of unions

Introduction

In this chapter, we will discuss

- Need of user-defined data type
- Structure definition
- Declaration and Initialization of variables
- Array as member
- Array of structure variables
- Structure within structure
- Structure as function arguments
- Union
- Bit fields

Need of user-defined data type

- To support database, file read/write, and print operations.
- A user-defined data type lets you group data of different types in a single variable.
- This data type can contain any kind of related information you want to store and use together, such as personnel information, company financial information, inventory, and customer and sales records.
- A variable of a user-defined data type holds actual data, not a pointer to that data.

Structure definition

- A structure can be considered as a template used for defining a collection of variables under a single name.

- Structures help programmers to group elements of different data types into a single logical unit (Unlike arrays which permit a programmer to group only elements of same data type).

- Structures are commonly used to define records to be stored in files Structures are very powerful concept that programmers use in many C programs.

Structure definition Continue…

- **Syntax**

```
struct tag {
        member 1;
        member 2;
        :
        member m;
    };
```

- **struct** is the required C keyword
- **tag** is the name of the structure
- **member 1, member 2,** … are individual member declarations

Structure definition Continue…

- The individual members can be ordinary variables, pointers, arrays, or other structures (any data type)
- The member names within a particular structure must be distinct from one another
- A member name can be the same as the name of a variable defined outside of the structure
- Once a structure has been defined, the individual structure-type variables can be declared as:

struct tag var_1, var_2, …, var_n;

Structure definition Continue…

A Compact Form Syntax :

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
        member 1;
        member 2;
        :
        member m;
} var_1, var_2,…, var_n;
```

- Declares three variables of type **struct tag**
- In this form, **tag** is optional

Structure definition Continue…

Various Structure declaration form :

```
struct S {
   int a;
   float b;
} x;
```

Declares x to be a structure having two members, a and b. In addition, the structure tag S is created for use in future declarations

```
struct {
int a;
float b;
} z;
```

Omitting the tag field;
cannot create any more variables with the same type as z

```
struct S {
int a;
float b;
};
```

Omitting the variable list defines the tag S for use in later declarations

Structure definition Continue…

```
struct S y;
```

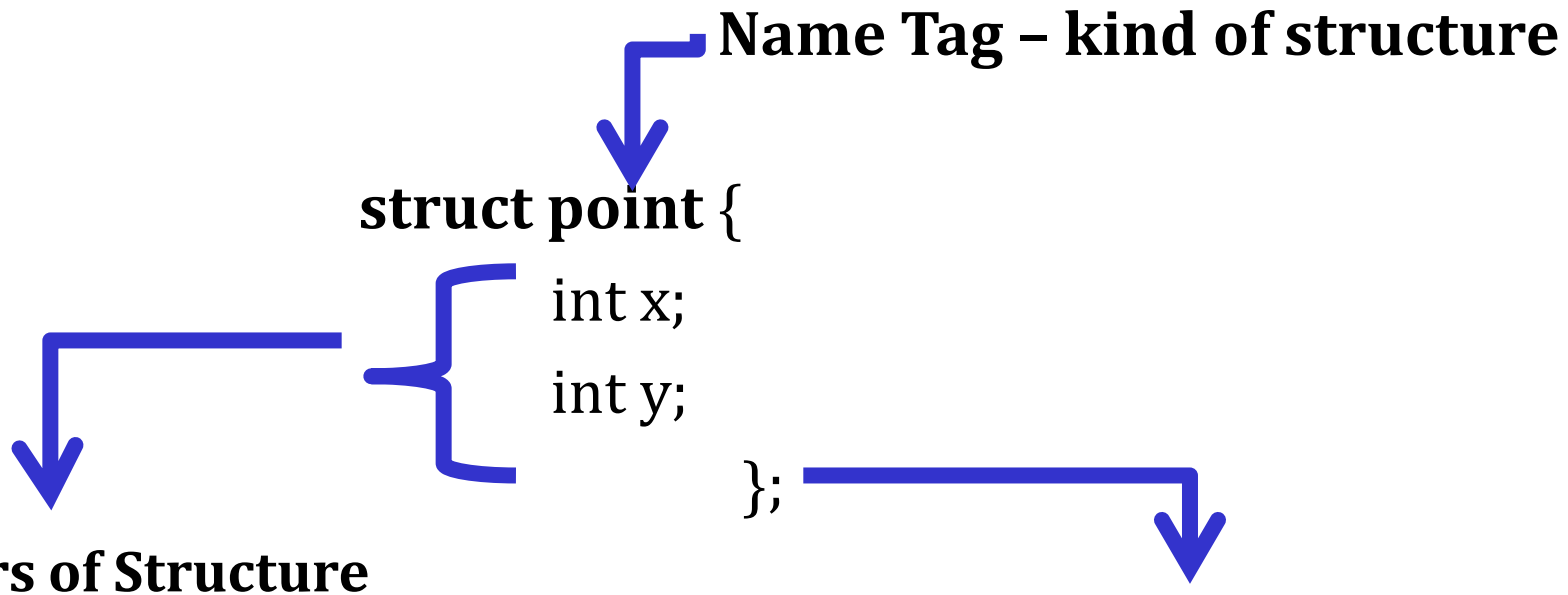
Omitting the member list declares another structure variable y with the same type as x omitting.

```
struct S;
```

Incomplete declaration which informs the compiler that S is a structure tag to be defined later .

Structure definition Continue…

- Example 1:
- In Cartesian coordinates any point can be described with two numbers corresponding *x*- and *y-coordinates.*



Structure definition Continue…

Note

This is just the structure definition. Structure definitions doesn't consume any space on the memory.

Structure definition Continue…

- Example 2:
- Let's suppose that we are dealing with **the employee records;** employee's record contains information such as employee's first and last names, employee's age, 'M' or 'F' for the employee's gender and the employee's hourly salary.

Employee Information:
First Name : Character array
Last Name : Character array
Age : Integer
Gender : A character
Hourly Salary : Double

Structure definition Continue…

- Example 2 Continued…

We can define a structure type with tag, employee

```
struct employee {
    char firstName[ 20 ];
    char lastName[ 20 ];
    int age;
    char gender;
    double hourlySalary;
};
```

Structure definition Continue…

Note

Members of a structure can be primitive data types and as well
as another structure.

Declaration and Initialization of variables

Different Ways of Declaring Structure Variable :

Way 1 : Immediately after Structure Template

```
struct date
{
      int date;
      char month[20];
      int year;
}today;

// 'today' is name of Structure variable
```

Declaration of variables Continue…

Way 2 : Declare Variables using struct Keyword

```
struct date
{
    int date;
    char month[20];
    int year;
};

struct date today;
```

where "**date**" is name of structure and "**today**" is name of variable.

Declaration of variables Continue…

Way 3 : Declaring Multiple Structure Variables

```
struct Book
{
    int pages;
    char name[20];
    int year;

}book1,book2,book3;
```

We can declare multiple variables separated by comma directly after closing curly.

Initialization of variables

- Different ways to initialize structure variable :

Way 1 : Declare and Initialize

```
struct student
{
    char name[20];
  int roll;
  float marks;
}std1 = { "Pritesh",67,78.3 };
```

In the code snippet, we have seen that structure is declared and as soon as after declaration we have initialized the structure variable.

std1 = { "Pritesh",67,78.3 }

Initialization of variables Continue…

Way 2 : Declaring and Initializing Multiple Variables

```
struct student
{
    char name[20];
    int roll;
    float marks;
}

std1 = {"Pritesh",67,78.3};
std2 = {"Don",62,71.3};
```

In this example, we have declared two structure variables in above code. After declaration of variable we have initialized two variable.

std1 = {"Pritesh",67,78.3};
std2 = {"Don",62,71.3};

Initialization of variables Continue…

Way 3 : Initializing Single member

```
struct student
{
    int mark1;
    int mark2;
    int mark3;
} sub1={67};
```

Though there are three members of structure, only one is initialized , Then remaining two members are initialized with Zero. If there are variables of other data type then their initial values will be -

Data Type	Default value if not initialized
integer	0
float	0.00
char	NULL

Initialization of variables Continue…

Way 4 : Initializing inside main

```
struct student
{
    int mark1;
    int mark2;
    int mark3;
};

void main()
{
struct student s1 = {89,54,65};
- - - - --
- - - - --
- - - - --
};
```

We need to initialize structure variable to allocate some memory to the structure.

Declaration and Initialization of variables Continue…

Note

When we declare a structure then memory won't be allocated for the structure, only writing declaration statement will never allocate memory

Accessing Structure variable

- So far, we have learned how to define a structure, but didn't talk about how to use it.
- In order to use structures, we should be able to access the members of the structure.
- Two operators are used to access members of structures:
- 1. The structure member operator (.) – also called dot operator.

  VariableName.MemberName
- 2. The structure pointer operator (->) – also called arrow operator.

  PointerName->MemberName
- *It is possible define a pointer to a structure. -> is used to access the member of the structure variable via pointers.*

Accessing Structure variable

Example : Accessing Structure variable using dot (.) Operator

- Let's now consider the date structure (mm/dd/yyyy)

```
struct date{                          /*Defining Structure Type*/
        int month;
        int day;
        int year;
        };
struct date today;
```

- Members of the today structure variables can be accessed via . operator.
- We can print, read, or assign any value to members by accessing
- them using . operator.

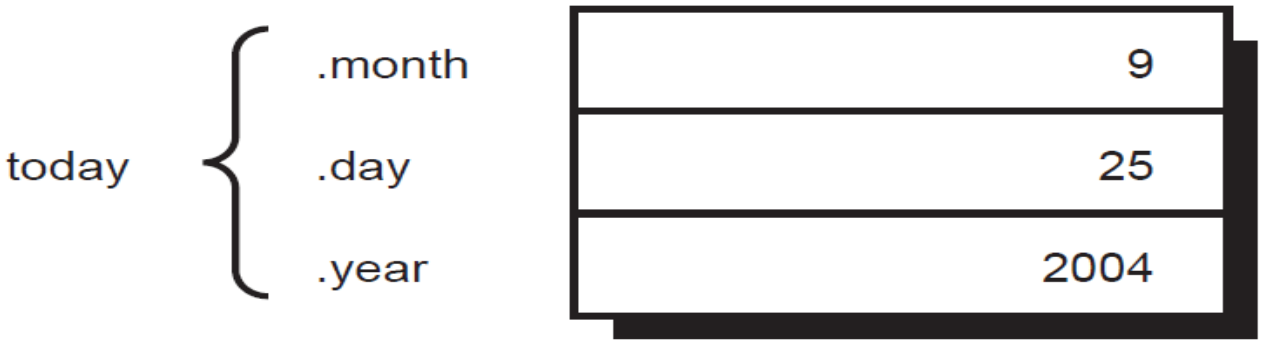
Accessing Structure variable

Example Continue…

- today.month = 9;
- today.day = 25;
- today.year = 2004;

- Below printf statement prints the date in (mm/dd/yyyy) format.

```
printf("%d/%d/%d", today.month, today.day, today.year);
```



Declaration and Initialization of variables Continue…

Some Important Points Regarding Structure in C Programming :

1. **Struct** keyword is used to declare structure.
2. **Members of structure** are enclosed within opening and closing braces.
3. **Declaration** of Structure reserves **no space**.
4. It is nothing but the " **Template / Map / Shape** " of the structure .
5. Memory is created , very first time when the **variable is created /Instance** is created.

Array as member

- We can use structure and array as :

  1. Array within Structure (Array as Member)
  2. Array of Structure

1. Array as Member :

- Structure may contain the Array as its Member.
- We can store name,roll,percent as structure members , but sometimes we need to store the marks of three subjects then we embed integer array of marks as structure member.
- Syntax :

```
struct struct-name
{
        datatype var1; // normal variable
        datatype array [size]; // array variable
        datatype varN;
};
struct struct-name obj;
```

Array as member

Example 1 :

```
struct student
{
        char sname[20];  // array variable
        int roll;
        float percent;
        int marks[3];     // array variable
}s1;
```

Array as member Continue…

- Example Continue…

```
#include <stdio.h>
struct student
{
char sname[20];
int marks[3]; //Note Carefully
}s1;
int main()
{
printf("\nEnter the Name of Student : ");
gets(s1.sname)
```

Array as member Continue…

- Example Continue…

```
printf("\nEnter the Marks in Subject 1 :");
scanf("%d",&s1.marks[0]);
printf("\nEnter the Marks in Subject 2 : ");
scanf("%d",&s1.marks[1]);
printf("\nEnter the Marks in Subject 3 : ");
scanf("%d",&s1.marks[2]);
printf("\n ---- Student Details -------- ");
printf("\n Name of Student : %s",s1.sname);
printf("\n Marks in Subject 1 : %d",s1.marks[0]);
printf("\n Marks in Subject 2 : %d",s1.marks[1]);
printf("\n Marks in Subject 3 : %d",s1.marks[2]);
return 0; }
```

Array as member Continue…

- **Example 2:** Create another structure called student to store name, roll no and marks of 5 subjects.

```
struct student
{
        char name[20];      // array variable
        int roll_no;
        float marks[5];     // array variable
};
```

- If **student_1** is a variable of type struct student then

  student_1.marks[0] - refers to the marks in the first subject
  student_1.marks[1] - refers to the marks in the second subject

- and so on..

Array of structure variables

2. Array of Structure variable

- An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an array of structure or object.
- As we know, an array is a collection of similar type, therefore an array can be of structure type.
- Syntax :

```
struct struct-name
{
        datatype var1;
        datatype var2;
        - - - - - - - - - -
        datatype varN;
};
struct struct-name obj [ size ];
```

Array of structure variables

Example 1 :

```
#include<stdio.h>
struct Employee
{
        int Id;
        char Name[25];
        int Age;
        long Salary;
};
void main()
{
        int i;
        struct Employee Emp[ 3 ]; //Statement 1
        for(i=0;i<3;i++)
        {
```

Array of structure variables.

Example Continue…

```
printf("\nEnter details of %d Employee",i+1);
printf("\n\tEnter Employee Id : ");
scanf("%d",&Emp[i].Id);
printf("\n\tEnter Employee Name : ");
scanf("%s",&Emp[i].Name);
printf("\n\tEnter Employee Age : ");
scanf("%d",&Emp[i].Age);
printf("\n\tEnter Employee Salary : ");
scanf("%ld",&Emp[i].Salary);      }
printf("\nDetails of Employees");
for(i=0;i<3;i++)
 printf("\n%d\t%s\t%d\t%ld",Emp[i].Id,Emp[i].Name,Emp[i].A
                                                    ge,Emp[i].Salary);
}
```

Array of structure variables

Example Explanation :

- In the above example, we are getting and displaying the data of 3 employee using array of object.

- **Statement 1** is creating an array of Employee Emp to store the records of 3 employees.

Structure within structure

- Structure written inside another structure is called as nesting of two structures.
- Nested Structures are allowed in C Programming Language.
- We can write one Structure inside another structure as **member** of another structure.
- **Syntax 1 : Nested Structure**

```
struct structure1
{
 - - - -
 - - - -
};
struct structure2
{
- - - -
- - - -
struct structure1 obj;
};
```

Structure Within structure Continue…

- **Syntax 2 : Embedded Structure**

```
struct structure1
{
 - - - -
 - - - -
struct structure2
{
- - - -
- - - -
}structure2 obj;
}structure1 obj;
```

Structure Within structure Continue…

Example 1 : **Way of declaration of nested structure**

```
struct date
{
        int date;
        int month;
        int year;
};
struct Employee
{
        char ename[20];
        int ssn;
        float salary;
        struct date doj;
} emp1;
```

Structure Within structure Continue…

Example 1 : **Way of Accessing Elements of Nested Structure**

- Structure members are accessed using **dot operator.**
- '**date**'structure is nested within Employee Structure.
- Members of the '**date**' can be accessed using 'employee'
- **emp1 & doj are two structure names (Variables)**

- **Explanation of Nested Structure :**

  Accessing Month Field: emp1.doj.month

  Accessing day Field : emp1.doj.day

  Accessing year Field: emp1.doj.year

Structure Within structure Continue…

Example 2 : **Way of declaration of embedded structures**

```
struct Employee
{
        char ename[20];
        int ssn;
        float salary;
        struct date
{
        int date;
        int month;
        int year;
}doj;
}emp1;
```

Structure Within structure Continue…

Example 2 : **Way of Accessing Elements of Nested Structure member**

Accessing Nested Members :

Accessing Month Field : emp1.doj.month

Accessing day Field : emp1.doj.day

Accessing year Field : emp1.doj.year

Structure Within structure Continue…

Program :

```
#include <stdio.h>
#include<conio.h>
struct Employee
{
        char ename[20];
        int ssn;
        float salary;
        struct date
        {
                int date;
                int month;
                int year;
        }doj;
}emp = {"Pritesh",1000,1000.50,{22,6,1990}};
```

Structure Within structure Continue…

```
Void main()
{
        printf("\nEmployee Name : %s",emp.ename);
        printf("\nEmployee SSN : %d",emp.ssn);
        printf("\nEmployee Salary : %f",emp.salary);
        printf("\nEmployee DOJ :
%d/%d/%d",emp.doj.date,emp.doj.month,emp.doj.year);
        getch();
}
```

Structure as function arguments

- **It is possible to pass a structure or structures to functions in two ways.**

1. Structures may be passed to functions by passing individual structure members **one by one (Call-by-Value)**.

In this method, structure members are passed by value, therefore, the members of a caller's structure cannot be modified by the called function.

2. It is also possible to pass **the entire structure** or by passing **a pointer to the structure (Call-by-Reference)**.

In this method, structure members are passed by reference (like memory address) and the caller's structure can be modified by the function (as like passing an array with reference).

Structure as function arguments Continue…

Passing structure by value (Call By Value)

- A structure variable can be passed to the function as an argument as normal variable.
- If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.
- Example :

```
#include <stdio.h>
struct student{
char name[50];
int roll;
};
void Display(struct student stu);
```

Structure as function arguments Continue…

```
int main(){
struct student s1;
printf("Enter student's name: ");
scanf("%s",&s1.name);
printf("Enter roll number:");
scanf("%d",&s1.roll);
Display(s1); // passing structure variable s1
                    as argument
return 0;
}
void Display(struct student stu){
printf("Output\nName: %s",stu.name);
printf("\nRoll: %d",stu.roll);
}
```

```
Enter student's name: Kevin Amla

Enter roll number: 149

Output

Name: Kevin Amla

Roll: 149
```

Output

Structure as function arguments Continue…

Passing structure by reference (Call By Reference)

- The address location of structure variable is passed to function while passing it by reference.
- If structure is passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.
- Example :

```
#include <stdio.h>
struct distance{
int feet;
float inch;
};
void Add(struct distance d1,struct distance d2, struct distance *d3);
```

Structure as function arguments Continue…

```
int main(){
struct distance dist1, dist2, dist3;
printf("First distance\n");
printf("Enter feet: ");
scanf("%d",&dist1.feet);
printf("Enter inch: ");
scanf("%f",&dist1.inch);
printf("Second distance\n");
printf("Enter feet: ");
scanf("%d",&dist2.feet);
printf("Enter inch: ");
scanf("%f",&dist2.inch);
Add(dist1, dist2, &dist3);
```

Structure as function arguments Continue…

```
printf("\nSum of distances = %d\'-%.1f\"",dist3.feet, dist3.inch);
return 0; }
void Add(struct distance d1,struct distance d2, struct distance *d3){
/* Adding distances d1 and d2 and storing it in d3 */
d3->feet=d1.feet+d2.feet;
d3->inch=d1.inch+d2.inch;
if (d3->inch>=12) { /* if inch is greater or equal to 12,
converting it to feet. */
d3->inch-=12;
++d3->feet;
} }
```

```
First distance

Enter feet: 12

Enter inch: 6.8

Second distance

Enter feet: 5

Enter inch: 7.5

Sum of distances = 18'-2.3"
```

Declaration and Initialization of variables Continue…

Note

Passing structure by reference is a lot more efficient than passing structures by value. Passing a structure variable to a function is very similar to passing an array to a function.

Union

- Unions are quite similar to the structures in C. Union is also a derived type as structure.
- Union can be defined in same manner as structures just the keyword used in defining union is **union** where keyword used in defining structure was **struct.**
- **Syntax :**

```
union car{
        char name[50];
        int price;
};
```

- Union variables can be created in similar manner as structure variable.

```
union car{
        char name[50];
        int price;
}c1, c2, *c3;
```

Union Continue…

Or Variable can be created inside function

```
union car{
        char name[50];
        int price;
};
-------Inside Function-----------
union car c1, c2, *c3;
```

Union Continue…

Accessing members of an union :

- The member of unions can be accessed in similar manner as that structure.

- Suppose, we you want to access price for union variable **c1** in above example, it can be accessed as **c1.price**.

- If you want to access price for union pointer variable **c3**, it can be accessed as **(*c3).price** or as**c3->price**.

Union Continue…

Difference between union and structure

1. There is difference in memory allocation between union and structure .

- The amount of memory required to store a structure variables is the sum of memory size of all members.
- But, the memory required to store a union variable is the memory required for largest element of an union.

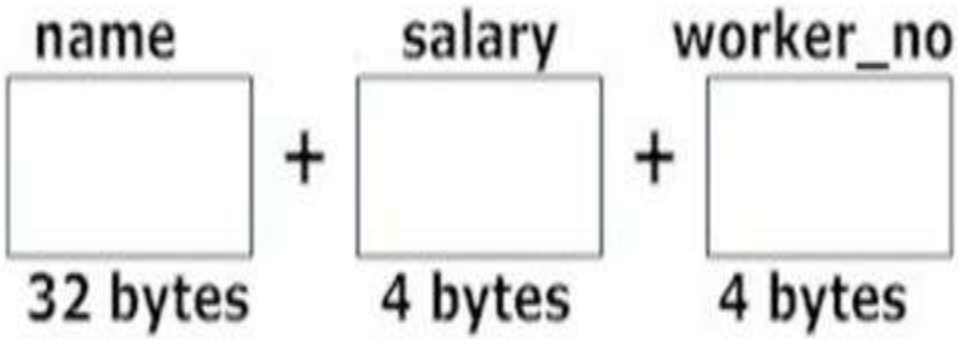


Fig: Memory allocation in case of structure

Fig: Memory allocation in case of union

Union Continue…

Difference between union and structure

2. All members of structure can be accessed at any time.

- But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.
- Example :

```
#include <stdio.h>
union job {
        char name[32];
        float salary;
        int worker_no;
}u;
int main(){
        printf("Enter name:\n");
        scanf("%s",&u.name);
        printf("Enter salary: \n");
        scanf("%f",&u.salary);
        printf("Displaying\nName :%s\n",u.name);
        printf("Salary: %.1f",u.salary);
        return 0;
}
```

Output

```
Enter name

Hillary

Enter salary

1234.23

Displaying
Name: f%Bary

Salary: 1234.2
```

Bit Field

- In addition to bitwise operators, we can declare structures whose members represent bit-fields

Bit Field Declaration

```
struct
{
        type [m em ber_nam e] : width ;
};
```

Below the description of variable elements of a bit field:

Elements Description :

type An integer type that determines how the bit-field's value is interpreted. The type may be int, signed int, unsigned int.

member_name The name of the bit-field.

width The number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type.

Bit Field Continue…

Example :

```
struct
{
        unsigned int widthValidated : 1;
        unsigned int heightValidated : 1;
} status;
```

- The above structure will require 4 bytes of memory space for status variable but only 2 bits will be used to store the values.

- If you will use up to 32 variables each one with a width of 1 bit , then also status structure will use 4 bytes, but as soon as you will have 33 variables, then it will allocate next slot of the memory and it will start using 8 bytes.

Bit Field Continue…

- Example :

```
#include <stdio.h>
#include <string.h>
/* define simple structure * /
struct
{
unsigned int widthValidated;
unsigned int heightValidated;
} status1;
/* define a structure with bit fields * /
struct
{
unsigned int widthValidated : 1;
unsigned int heightValidated : 1
} status2;
```

Bit Field Continue…

```
int main( )
{
printf( "Mem ory size occupied by status1 : %d\n", sizeof(status1));
printf( "Mem ory size occupied by status2 : %d\n", sizeof(status2));
return 0;
}
```

Output

```
Memory size occupied by status1 : 8
Memory size occupied by status2 : 4
```

Previous Year Question

Fill in the blanks

1. Structure is a ____ data type.
2. Memory is allocated for a structure when ___ is done.
3. A structure member variable is generally accessed using a ___.
4. ____ is a collection of data under one name in which memory is shared among the members.
5. A structure placed within another structure is called a ___.
6. The selection operator is used to _____.

Previous Year Question

Answer the following MCQ :

1. A Data structure that can store related information together is

a. Array **b. String**

c. Strucure **d. All of these**

2. A union Member variable is generally accessed using the

a. Address operator **b.dot operator**

c. Comma operator **d.ternary operator**

3.Typedef can be used with which of these data types?

a.Struct **b.union**

c.Enum **d.all of these**

Previous Year Question

State True or False :

1. A struct type is primitive data type.

2. Memory is allocated for a structure only when we declare variable of the structure.

3.Union and structure are initialized in the same way.

4. A structure cannot have union as its member .

5. C permits nested union.

6.No two members of a union should have the same name .

Presentation Prepared By:

Ms. Krishna Patel

Subject Teacher
Mr. Dipak Ramoliya

Contact us:

dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
nidhibarodwala.ce@charusat.ac.in
neetijoshi.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
phenilbuch.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal

Mr. Phenil Buch

Ms. Nidhi Barodawala

Ms. Neeti Joshi